OPTIMAL PARALLEL CONSTRUCTION OF PRESCRIBED TOURNAMENTS

Danny Soroker*

Computer Science Division
University of California
Berkeley, CA 94720

ABSTRACT

A *tournament* is a digraph in which every pair of vertices is connected by exactly one arc. The score list of a tournament is the sorted list of the out-degrees of its vertices. Given a non-decreasing sequence of non-negative integers, is it the score list of some tournament? There is a simple test for answering this question. There is also a simple sequential algorithm for constructing a tournament with a given score list. However, this algorithm has a greedy nature, and seems hard to parallelize. We present a simple parallel algorithm for the construction problem. Our algorithm runs in time $O(\log n)$ and uses $O(n^2/\log n)$ processors on a CREW PRAM, where n is the number of vertices. Since the size of the output is $\Theta(n^2)$, our algorithm achieves optimal speedup.

* Research supported by Defense Advanced Research Projects Agency (DoD) Arpa Order No. 4871, Monitored by Space & Naval Warfare Systems Command under Contract No. N00039-84-C-0089 and by the International Computer Science Institute, Berkeley, California

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **SEP 1987**	2. REPORT TYPE	3. DATES COVERED **00-00-1987 to 00-00-1987**

4. TITLE AND SUBTITLE **Optimal Parallel Construction of Prescribed Tournaments**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
http://www.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-371.pdf A tournament is a digraph in which every pair of vertices is connected by exactly one arc. The score list of a tournament is the sorted list of the out-degrees of its vertices. Given a non-decreasing sequence of non-negative integers, is it the score list of some tournament? There is a simple test for answering this question. There is also a simple sequential algorithm for constructing a tournament with a given score list. However, this algorithm has a greedy nature, and seems hard to parallelize. We present a simple parallel algorithm for the construction problems. Our algorithm runs in time O(log n) and uses O(n^2/logn) processors on a CREW PRAM, where n is the number of vertices. Since the size of the output is Omega(n^2), our algorithm achieves optimal speedup.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**12**	

1. Introduction

A *tournament* is a directed graph in which there is exactly one arc between every pair of vertices. This models a competition involving n players, where every player competes against every other one. If an arc is directed from x to y we say that x *dominates* y. The *transitive tournament* on n vertices is the tournament in which each integer between 1 and n has a corresponding vertex, and i dominates j if $i>j$. The *score* of a vertex is the number of vertices it dominates. The *score list* of a tournament is the sorted list of scores of its vertices (starting with the lowest).

Tournaments are widely studied in the literature (e.g. [BW],[M]). In this paper we deal with the following problem: given a non-decreasing list of integers, $\vec{s}=s_1, \ldots, s_n$, determine if there exists a tournament with score list $\vec{s}$, and if so, construct such a tournament.

A simple, non-constructive criterion for testing if such a list is a score list was found by Landau in 1953 ([BW]): $\vec{s}$ is a score list of some tournament if and only if, for all k, $1 \le k \le n$:

$$\sum_{i=1}^{k} s_i \ge \binom{k}{2}$$

with equality for $k=n$.

A simple greedy algorithm ([BW,CL]) is known for constructing a tournament with v_i having score s_i (for all $1 \le i \le n$): select some score s_i and remove it from the list; have v_i dominate the s_i vertices with smallest scores (and have the rest of the vertices dominate v_i); subtract 1 from the score of each vertex dominating v_i and repeat this procedure for the reduced list. We note that very similar algorithms exist for several other construction problems ([B],[CL],[FF]).

The main result of this paper is an NC algorithm for the construction problem. Our algorithm runs in time $O(\log n)$ and uses $O(n^2/\log n)$ processors on a concurrent read - exclusive write (CREW) PRAM, where n is the number of vertices. Since the size of the output is $\Theta(n^2)$, our algorithm achieves optimal speedup. See e.g. [V] for a discussion of parallel algorithms and optimal speedup.

In section 2 we describe our approach, which is based on looking at arcs that go from vertices of lower score to vertices of higher score.

In section 3 we derive a high-level description of the parallel algorithm.

Section 4 contains a detailed description of an implementation of the algorithms that achieves optimal speedup.

2. The Upset Sequence

Our approach is based on, what we call, the **upset sequence** of a tournament, T, which describes the difference between T and a transitive tournament. If we list the vertices according to their scores in non-decreasing order, then an *upset* is when a vertex, v, dominates some other vertex appearing later than v in the list. We call an arc corresponding to an upset a *reverse arc.* Transitive tournaments are exactly those tournaments that contain no upsets.

Definition: Let $s_1 \leq \cdots \leq s_n$ be the score list of a tournament, T, and let v_i be the vertex of score s_i (for all $1 \leq i \leq n$). The *upset sequence* of T, is the sequence, $\vec{u}$, where u_k is the number of upsets between $\{v_1, \ldots, v_k\}$ and $\{v_{k+1}, \ldots, v_n\}$ (for all $1 \leq k \leq n-1$).

The score list uniquely determines the upset sequence (and vice-versa):

Lemma 2.1: Let T be a tournament with score list $\vec{s}$ and upset sequence $\vec{u}$. Then, for all $0 \leq k \leq n-1$:

$$u_k = \sum_{i=1}^{k}(s_i - i + 1) = \sum_{i=1}^{k} s_i - \binom{k}{2}$$

Proof: There are exactly $\binom{k}{2}$ arcs in the subgraph induced on $\{v_1, \ldots, v_k\}$, since it is also a tournament. Therefore the right hand side describes the number of arcs whose tail is in $\{v_1, \ldots, v_k\}$, but whose head isn't. $\square$

Corollary 2.1: For all $1 \leq k \leq n$:

$$s_k = u_k - u_{k-1} + k - 1$$

How can we use the upset sequence? Our approach is to construct a tournament with a given score list by starting with a transitive tournament and reversing some of its arcs. The upset sequence of the desired tournament gives us a handle on which arcs to reverse. We will be aided by a graphical representation of the upset sequence, which we now discuss.

A sequence of non-negative integers can be represented graphically by its histogram. We will treat the histogram as a rectilinear polygon (and call it, simply, a *polygon*), which is divided into squares, each of which has integral x and y coordinates. The x coordinate is a square's *column* and the y coordinate is its *height.* An example of a polygon is shown in fig. 2.1. Any collection of squares of a polygon is a *sub-polygon.* A maximal set of consecutive squares at the same height is called a *slice.* Note that a polygon can have several slices at the same height (if it is not convex). A (horizontal) *segment* is consecutive set of squares, all in the same slice. We denote a segment or slice by $[l,r]$ or by $[l,r;h]$, where l and r are, respectively, the columns of the leftmost and rightmost squares it contains, and h is its height.

A polygon representing the upset sequence of a tournament will be called an *upset polygon.*

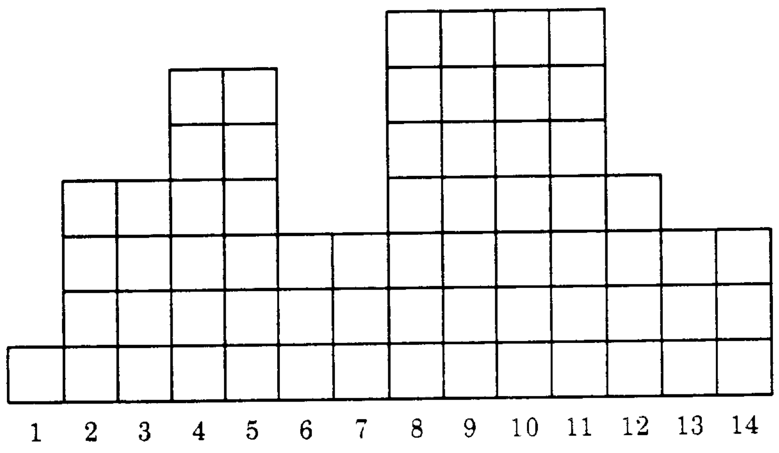


Fig. 2.1: A polygon representing the sequence 1,4,4,6,6,3,3,7,7,7,7,4,3,3.

An elementary property of a polygon, which follows from its definition is:
Proposition 2.1: The slices of a polygon form a nested structure: if $[l_1,r_1]$ and $[l_2,r_2]$ are slices with $l_1 \geq l_2$ then either $l_1 > r_2$ or $r_1 \leq r_2$.

We define the following partitioning problem: Given a rectilinear polygon as shown in fig. 2.1, partition each of its slices into segments such that no two segments in the partition agree on both endpoints. Such a partition is said to be *valid*, and is defined by the set of segments it contains. An example of a valid partition is illustrated in fig. 2.2. The partition is {[1,14], [2,4], [2,5], [2,14], [4,4], [4,5], [5,5], [5,14], [8,8], [8,9], [8,10], [8,11], [9,11], [10,11], [11,12]}.

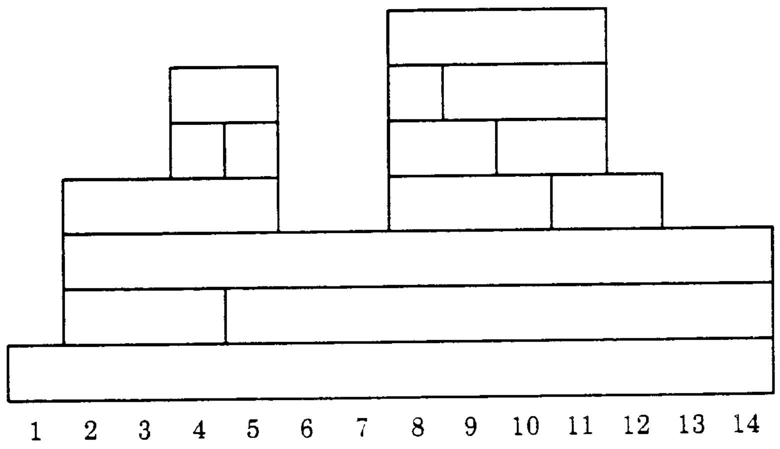


Fig. 2.2: A valid partition of the polygon of fig. 2.1.

Lemma 2.2: A valid partition of the upset polygon yields a solution to the construction problem.

Proof: Let $\{[l_i,r_i] \mid 1 \le i \le m\}$ be the set of segments in a valid partition of an upset polygon representing a sequence $\vec{u}$ corresponding to the score list $\vec{s} = s_1, \ldots, s_n$. Let T be the tournament obtained by taking the n vertex transitive tournament and reversing the arcs $\{(r_i,l_i) \mid 1 \le i \le m\}$. By inspection, the number of reverse arcs crossing the cut $(\{v_1, \ldots, v_k\}:\{v_{k+1}, \ldots, v_n\})$ is exactly u_k. Therefore (by corollary 2.1), T is a tournament with score list $\vec{s}$. []

Note that each slice in fig. 2.2 is partitioned into at most two segments. This is not a coincidence.

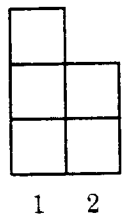
Definition: A <u>*2−partition*</u> is a valid partition in which every slice is partitioned into at most 2 segments. A slice which is partitioned into at most 2 segments is <u>*2−partitioned.*</u>

We will deal only with 2-partitions because of the following:

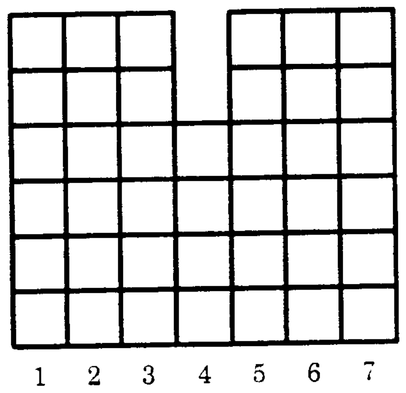
Lemma 2.3: If a polygon has a valid partition, then it has a 2-partition.
Proof: Let P be a valid partition of some polygon, which is not a 2-partition. Let S be a slice which is partitioned into more than 2 segments such that all slices lying above S are 2-partitioned. We will prove the lemma by showing how to transform P into another valid partition in which S is 2-partitioned and the partition of slices above S is unchanged.

Let the segments comprising S in P be, from left to right, $[l_1,r_1], \ldots, [l_k,r_k]$ (where $k>2$). If either $[l_1,r_{k-1}]$ or $[l_2,r_k]$ does not appear in P, then the partition of S can be replaced with $\{[l_1,r_{k-1}],[l_k,r_k]\}$ or $\{[l_1,r_1],[l_2,r_k]\}$ respectively. If both appear, then at least one, say $[l_1,r_{k-1}]$, must appear in a slice below S (call this slice T). This follows from the assumption that all slices lying above S are 2-partitioned and from the nesting property (proposition 2.1). Now, simply assign the segment $[l_1,r_{k-1}]$ to S and the segments $[l_1,r_1], \ldots, [l_k,r_k]$ to T. []

Not every rectilinear polygon of the type discussed has a valid partition. Two examples are shown in fig 2.3.



(1)



(2)

Fig. 2.3: Examples of polygons which have no valid partition.

We will show, however, that every upset polygon has a 2-partition. A few more definitions are required for this: a _left (right) face_ is a maximal vertical line segment on the left (right) part of the boundary of a polygon. _Face k,_ if it exists, is the face between columns $k-1$ and k. Two faces, L and R, are _opposing_ if there is some slice starting at L and ending at R. The _width, $w(F)$_ of a face, F, is the *minimum* distance between it and any of its opposing faces (where distance is measured by number of squares). The length of a face F (i.e the number of slices it touches) is denoted by $l(F)$.

Lemma 2.4: A polygon, D, has a 2-partition if the length of every face of D is no more than half its width.

Proof: We prove the lemma by induction on the height of D. If the height is 1 then D clearly has a 2-partition. Assume the claim holds for all polygons of height $k-1$, and let k be the height of D. Let D' be the polygon obtained by removing the bottom slice from D. By the inductive assumption, D' has a 2-partition, P. We will show that P can be extended to a 2-partition of D. Let L and R be, respectively, the left and right faces bounding the bottom level of D. P contains $l(L)-1$ segments starting at L and $l(R)-1$ segments ending at R. By the condition of the lemma,

$$\textit{width of bottom slice} \geq w(L)\ ,\ w(R) \geq l(L)+l(R)$$

Therefore, by the pigeonhole principle, there are two segments that partition the bottom slice, which are not contained in P. Thus P can be extended to become a 2-partition of D. []

Lemma 2.5: In an upset polygon the length of every face is no more than half its width.

Proof: Let $\Delta(k)$ be the difference in height between the highest square with x-coordinate k and the highest square with x-coordinate $k-1$. In other words, if F is a left face bounding squares with x-coordinate k, then $\Delta(k)=l(F)$. If F is a right face then $\Delta(k)=-l(F)$. Using corollary 2.1:

$$\Delta(k) = u_k - u_{k-1} = s_k - k + 1$$

Since $\vec{s}$ is non-decreasing, it follows that:

$$(*) \qquad \textit{for all } 2 \le k \le n-1 \qquad \Delta(k) \ge \Delta(k-1)-1$$

Say face k is a left face, L. The nearest opposing face of L occurs to the right of the first value, r, such that $r>k$ and $\Delta_{k+1}+\Delta_{k+2}+ \cdots +\Delta_r<0$. The smallest possible r value can occur (by (*)) when $\Delta_k=\Delta_{k+1}+1= \cdots =\Delta_r+r-k$. In this case:

$$w(L) = r-k+1 = 2\Delta(k) = 2l(L)$$

A symmetric argument works for right faces. []

Theorem 2.1: Every upset polygon has a 2-partition.

3. 2-Partitioning the Upset Polygon

As described in the previous section, our algorithm works as follows: given the score list, $\vec{s}$, we compute its corresponding upset sequence $\vec{u}$ and construct a 2-partition, P, of the upset polygon. In the output tournament, for all $1 \le i < j \le n$, v_i dominates v_j if and only if $[i,j] \in P$.

What remains to be shown is how to compute a 2-partition of an upset polygon, U, efficiently in parallel. Basically, our approach is to construct the partition according to faces. We first observe that it is a simple task to partition a set of slices with a common face as follows: say the common face is a left face. Let the set of slices be, from top to bottom, $S_1, \ldots, S_m$, where $S_i=[l,r_i]$ for all $1 \le i \le m$. Then S_i will be partitioned into the segments $[l,l+i]$ and $[l+i+1,r_i]$. This is shown in fig. 3.1. Such a partition is always possible given lemma 2.5. A symmetric partition exists for slices sharing a right face.

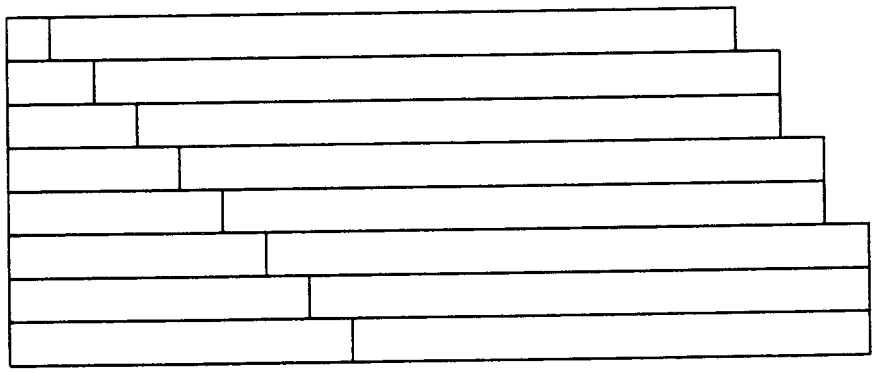


Fig. 3.1: 2-partitioning a set of slices with a common left face.

If we simultaneously partition the entire polygon in the manner described (according to left faces), the resulting partition might not be valid, since a right face can be opposite several left faces. Our solution is to have every slice "belong" to one of (the two) faces it touches, and to be partitioned accordingly. More specifically, it belongs to the dominant face according a *domination* relationship defined as follows: a left face, L, dominates an opposing right face, R, unless the top slice touching L touches R but the top slice touching R *does not* touch L (in other words, R is the highest face opposing L but not vice-versa).

Theorem 3.1: Let $S=[l,r,h]$ be a slice belonging to face F. Let $S_F=[l',r',h']$ be the highest slice belonging to F. Say we partition S into 2 segments such that the length of the segment touching F is $h'-h+1$. If we perform this partitioning for all the slices of an upset polygon, U, then the result is a (valid) 2-partition of U.

Proof: First we note that if two slices belong to the same face, they must be of different height, so their partition cannot conflict (i.e. create segments with identical endpoints). Therefore, the only conceivable way in which a conflict can occur is from partitioning two slices, S_1 and S_2, that belong to faces L_1 and R_2 respectively, where L_1 and R_2 are left and right faces. Furthermore, L_1 and R_2 must be opposing faces because of the nesting property (proposition 2.1).

We note that the set of slices belonging to some face is consecutive. Say L_1 dominates R_2 (the other case is symmetrical). Then the right endpoint of a segment created from a slice belonging to L_1 is at distance at most $l(L_1)$ from L_1 and the left endpoint of a segment created from a slice belonging to R_2 is at distance at most $l(R_2)-1$ from R_2. Now we apply lemma 2.5: the distance between L_1 and R_2 is at least $l(L_1)+l(R_2)$. Therefore all right endpoints of segments created from slices belonging to L_1 are less than all left endpoints of segments created from slices belonging to R_2, so no conflict can occur. []

4. Implementation Details

We now describe in detail a parallel implementation of the tournament construction algorithm described above. Our algorithm works in time $O(\log n)$ and uses $O(n^2/\log n)$ processors on a concurrent read - exclusive write (CREW) PRAM, where n is the number of vertices in the tournament. Our parallel algorithm is **optimal**, since the size of the output is $\Theta(n^2)$. Some of the procedures will be easier to describe as using $O(n^2)$ processors and working in constant time. Each such procedure can clearly be slowed down to work in time $O(\log n)$ using only $O(n^2/\log n)$ processors.

Let U be the upset polygon corresponding to the input score list. The area of U (i.e. the number of squares it contains) can be $\Theta(n^3)$, since its height can be $\Theta(n^2)$ (for example, the area of an upset polygon of a a regular tournament is $\frac{(n-1)n(n+1)}{12}$). The first step we perform is to "compress" U to get an $O(n^2)$ representation.

Let $l_1 < l_2 < \cdots < l_m$ be the sorted list of values of the upset sequence $\vec{u}$ (l_i is the i'th smallest u value). The <u>*i'th level*</u> of U is the sub-polygon with y-coordinates between $l_{i-1}+1$ and l_i (where $l_0 \equiv 0$). It is easy to see that each level is a collection of *rectangles*. In other words, for every column j and level r, squares in j appear either in all the heights of r or in none of them. We can, thus, talk about "slices at level r". We represent U by a zero-one matrix, *LEVEL*, where $LEVEL[r,j]=1$ if and only if $u_j \geq l_r$. For a complete description we also keep a vector *HEIGHT*, where $HEIGHT[r]$ is the height of the highest slice in level r. *LEVEL* can be computed using $O(n^2)$ processors, each computing one entry in constant time.

We now list the steps of the computation. In each step a matrix or vector is computed, and in the final step a processor is assigned to each slice and 2-partitions it. We start by listing the matrices and vectors computed and then describe in detail how each step is implemented.

A vector *TOP_LEVEL*

$TOP\_LEVEL[k]$ is the maximum level, r, such that $LEVEL[r,k]=1$ (i.e. the highest level of column k).

A matrix *ENDPOINT*.

If there is a slice $[i,j]$ in level r, then $ENDPOINT[r,j]=i$ and $ENDPOINT[r,i]=j$. If no slice begins or ends at column j in level r then $ENDPOINT[r,j]=\Phi$.

Matrices *TOP* and *BOTTOM*.

$TOP[i,j]$ is the top level in which slice $[i,j]$ appears. $BOTTOM[i,j]$ is the bottom level in which slice $[i,j]$ appears. (Again, an entry is Φ if no such slice exists).

Face domination matrix, *FD*.

$FD[i,j]=1$ if face j dominates face i. $FD[i,j]=0$ if face i dominates face j. $FD[i,j]=\Phi$ if faces i and j are not opposing. (See section 3 for the definition of domination.)

Vector *TOP_SLICE*.

$TOP\_SLICE[k]$ is the level of the highest slice that belongs to face k (the face between columns $k-1$ and k).

TOP_LEVEL can be computed in constant time by assigning a processor to each entry of *LEVEL* to check if it is 1 and the entry above it is 0.

The r'th row of *ENDPOINT* is computed using $O(n/\log n)$ processors in $O(\log n)$ time by a balanced binary tree computation ([MR]). We "plant" a balanced complete binary tree with $n-1$ leaves on level r of the upset polygon. Each node, N, in the tree represents a range of entries in row r of *LEVEL*, between columns $l(N)$ and $r(N)$. A node computes three functions:

$propagate(N)$ - is **true** iff all the entries represented by N are 1.

$start\_right(N)$ - the first column of a slice starting between $l(N)$ and $r(N)$ and ending to the right of $r(N)-1$.

$end\_left(N)$ - the last column of a slice ending between $l(N)$ and $r(N)$ and starting to the left of $l(N)+1$.

An internal node, N, has two children, N_{left} and N_{right}, where $l(N_{left})=l(N)$, $r(N_{right})=r(N)$ and $r(N_{left})=l(N_{right})-1$. Then we have:

$propagate(N)$ = $propagate(N_{left})$ **and** $propagate(N_{right})$.
$start\_right(N)$ = **if** $propagate(N_{right})$ **then** $start\_right(N_{left})$ **else** $start\_right(N_{right})$.
$end\_left(N)$ = **if** $propagate(N_{left})$ **then** $end\_left(N_{right})$ **else** $end\_left(N_{left})$.

The leaves of the tree represent single entries. If an entry is 0 then $propagate$ = **false** and end_left and $start\_right$ are both Φ. If an entry is 1 then $propagate$ = **true** and end_left and $start\_right$ are both j (for the leaf representing entry j). A node computes its functions after its children have computed theirs. Furthermore, N, writes $end\_left(N_{right})$ in $ENDPOINT[start\_right(N_{left})$ and $start\_right(N_{left})$ in $ENDPOINT[end\_left(N_{right})$. Note that a value may be overwritten several times. After completing computing the functions for the whole tree, for each entry, j, if $LEVEL[r,j-1]=1$ and $LEVEL[r,j+1]=1$, then $ENDPOINT[r,j]$ is set to Φ.

It takes $O(\log n)$ time for the node functions to be evaluated for the entire tree. The whole computation can be done with $O(n/\log n)$ processors by a standard load-balancing trick, as described in [MR]. Proof that this procedure works correctly is straightforward, and is omitted.

TOP and *BOTTOM* are computed by having a processor for each entry of *ENDPOINT*. Processor [r,i] writes "j" in $TOP[i,j]$ if $ENDPOINT[r,i]=j$ and $ENDPOINT[r+1,i] \neq j$. Similarly for *BOTTOM*.

$FD[i,j]=1$ if $ENDPOINT[TOP[i,j]+1,j]=\Phi$ and either $ENDPOINT[TOP[i,j]+1,j] \neq \Phi$ or $i<j$).

For computing *TOP_SLICE*, let $t=ENDPOINT[TOP\_LEVEL[k],k]$. Then $[k,t]$ is the highest slice touching face k. If $FD[k,t]=1$ then $TOP\_SLICE[k]$ is equal to $TOP\_LEVEL[k]$. Otherwise, it is one level below $BOTTOM[k,t]$ (unless face k has no other slices than $[k,t]$, which can be checked by looking up $LEVEL[BOTTOM[k,t]-1,k]$).

Finally we partition each of the slices. Let $s=[l,r;h]$ be a slice. We use *FD* to find if s belongs to face l or face r. Then we use *TOP_SLICE* and *HEIGHT* to find the height, h', of the highest slice belonging to that face. Now we can partition s according to its height, h, and h' as described in theorem 3.1.

We need to show how to assign processors to slices. One way to do it is as follows: a vector, V, is created with one entry for each left face, with the entry being the length of the face. A vector, P, of partial sums of V is computed. This vector contains, essentially, an enumeration of the slices. Let α be the total number of slices of U. We assign $\log n$ consecutive slices to each of $\alpha/\log n$ processors. Each processor finds its first slice in time $O(\log n)$ by a binary search on P. After that, each of the successive slices is accessed in constant time.

Acknowledgments

Thanks to Richard Karp, Sampath Kannan and Noam Nisan for several helpful discussions.

References

[B] Berge, C. , "Graphs"
North Holland, 1985.

[BW]Beineke, L.W. and Wilson, R.S. eds. , "Selected Topics in Graph Theory"
Academic Press, 1978.

[CL] Chartrand, G. and Lesniak, L. , "Graphs & Digraphs" 2nd ed.
Wadsworth & Brooks/Cole , 1986.

[FF] Ford, L.R. and Fulkerson, D.R. , "Flows in Networks"
Princeton University Press, 1962.

[M] Moon, J.W. , "Topics on Tournaments"
Holt, Reinhart & Winston , 1968.

[MR]Miller, G.L. and Reif, J.H. , "Parallel Tree Contraction and its Application"
26'th FOCS, pp. 478-489, 1985.

[V] Vishkin, U., "Synchronous Parallel Communication - a Survey"
TR 71, Dept. of Computer Science, Courant Institute, NYU, 1983.